Model Based Verification of Cyber Range Event Environments

Suresh K. Damodaran
MIT Lincoln Laboratory
244 Wood St.,
Lexington, MA, USA
Suresh.Damodaran@ll.mit.edu

David Tidmarsh
MIT Lincoln Laboratory
244 Wood St.,
Lexington, MA, USA
David.Tidmarsh@ll.mit.edu

ABSTRACT

We apply model based verification to cyber range event environment configurations, allowing for the early detection of errors in event environment configurations, and a reduction in the time and resources used during deployment. We categorize misconfiguration errors detected using the Common Cyber Environment Representation (CCER) ontology. We also provide an overview of a methodology to specify verification rules and the corresponding error messages. These rules have successfully detected errors in the designs of several cyber range event environments, thereby reducing cost and time to deployment.

Author Keywords

Model based verification, Configuration, , Event environment, Verification rules, Error messages

ACM Classification Keywords

I.6.1 [SIMULATION AND MODELING]: Simulation Theory; I.6.5 [SIMULATION AND MODELING]: Model Development

1. INTRODUCTION

Cyber ranges provide the infrastructure needed to conduct testing, training, and exercise events over operationally relevant and representative environments. Conducting large cyber range events requires considerable effort and time. A cyber range event has four distinct phases: planning and design, deployment, execution, and post-execution [1]. One way to reduce the time taken to conduct an event is to reduce the time spent in the deployment phase. Improving procedures in the deployment phase also reduces the total amount of resources used, because some of the cyber range's most expensive resources are used during the deployment and execution phases. Any errors detected during deployment will require debugging and, therefore, add to the time and range resources used for deployment. This problem is not unique to cyber ranges. Configuration errors have been found to be one of the dominant causes of system failures [2]. It has been estimated that configuration errors account for 50% to 80% of the downtime and vulnerabilities in cyber infrastructure [3]. Therefore, finding and eliminating errors in the environment configuration prior to deployment can help diminish the time and resources consumed detecting, debugging, and fixing these errors during the event's deployment phase.

One effective way to reduce these errors is to verify and validate the environment configuration even before the environment is fully deployed. We explore the use of a model based approach for detection of configuration errors in the planning and design phase. The verification and validation of network configurations has been studied extensively [4,5,6,7,8,9]. However, previous work either does not provide a methodology for identifying and specifying rules, or describe how the corresponding rule violation error messages are generated to help environment designers. Both of these are important when verifying an event configuration that may include users, applications, operating systems, servers, hosts, routers, switches, control planes, and instrumentation planes, many of which lack models for their configuration.

Our main contributions in this paper are the following. First, we have developed a configuration ontology called Common Cyber Environment Representation (CCER). Second, we present a classification scheme for misconfiguration errors based on an analysis of the more than 125 rules currently implemented using CCER. Third, we establish the importance of visual elements in a diagram in generating error messages corresponding to these rules when the environments are described in diagrams. Finally, we describe a methodology for identifying and specifying verification rules and error messages for cyber range event environments. In the past two years, these rules have been used to flag configuration errors in environment designs for several cyber range events.

The rest of the paper is organized as follows. Section 2 provides an overview of the CCER ontology, and its current component ontologies. Section 3 describes the connections between diagrams, verification rules, and error messages, with examples from CCER. Section 4 provides a classification of misconfiguration errors detected using CCER. Section 5 provides an overview of a methodology for developing the verification rules, and for specifying corresponding rule violation error messages. Section 6 provides a case study motivating our claims about time savings in deployed ranges, Section 7 reviews related work, and Section 8 concludes this paper.

Model Based Verification of Cyber Range Event Environments

Suresh K. Damodaran
MIT Lincoln Laboratory
244 Wood St.,
Lexington, MA, USA
Suresh.Damodaran@ll.mit.edu

David Tidmarsh
MIT Lincoln Laboratory
244 Wood St.,
Lexington, MA, USA
David.Tidmarsh@ll.mit.edu

ABSTRACT
We apply model based verification to cyber range event environment configurations, allowing for the early detection of errors in event environment configurations, and a reduction in the time and resources used during deployment. We categorize misconfiguration errors detected using the Common Cyber Environment Representation (CCER) ontology. We also provide an overview of a methodology to specify verification rules and the corresponding error messages. These rules have successfully detected errors in the designs of several cyber range event environments, thereby reducing cost and time to deployment.

Author Keywords
Model based verification, Configuration, , Event environment, Verification rules, Error messages

ACM Classification Keywords
I.6.1 [SIMULATION AND MODELING]: Simulation Theory; I.6.5 [SIMULATION AND MODELING]: Model Development

1. INTRODUCTION

Cyber ranges provide the infrastructure needed to conduct testing, training, and exercise events over operationally relevant and representative environments. Conducting large cyber range events requires considerable effort and time. A cyber range event has four distinct phases: planning and design, deployment, execution, and post-execution [1]. One way to reduce the time taken to conduct an event is to reduce the time spent in the deployment phase. Improving procedures in the deployment phase also reduces the total amount of resources used, because some of the cyber range's most expensive resources are used during the deployment and execution phases. Any errors detected during deployment will require debugging and, therefore, add to the time and range resources used for deployment. This problem is not unique to cyber ranges. Configuration errors have been found to be one of the dominant causes of system failures [2]. It has been estimated that configuration errors account for 50% to 80% of the downtime and vulnerabilities in cyber infrastructure [3]. Therefore, finding and eliminating errors in the environment configuration prior to deployment can help diminish the time and resources consumed detecting, debugging, and fixing these errors during the event's deployment phase.

One effective way to reduce these errors is to verify and validate the environment configuration even before the environment is fully deployed. We explore the use of a model based approach for detection of configuration errors in the planning and design phase. The verification and validation of network configurations has been studied extensively [4,5,6,7,8,9]. However, previous work either does not provide a methodology for identifying and specifying rules, or describe how the corresponding rule violation error messages are generated to help environment designers. Both of these are important when verifying an event configuration that may include users, applications, operating systems, servers, hosts, routers, switches, control planes, and instrumentation planes, many of which lack models for their configuration.

Our main contributions in this paper are the following. First, we have developed a configuration ontology called Common Cyber Environment Representation (CCER). Second, we present a classification scheme for misconfiguration errors based on an analysis of the more than 125 rules currently implemented using CCER. Third, we establish the importance of visual elements in a diagram in generating error messages corresponding to these rules when the environments are described in diagrams. Finally, we describe a methodology for identifying and specifying verification rules and error messages for cyber range event environments. In the past two years, these rules have been used to flag configuration errors in environment designs for several cyber range events.

The rest of the paper is organized as follows. Section 2 provides an overview of the CCER ontology, and its current component ontologies. Section 3 describes the connections between diagrams, verification rules, and error messages, with examples from CCER. Section 4 provides a classification of misconfiguration errors detected using CCER. Section 5 provides an overview of a methodology for developing the verification rules, and for specifying corresponding rule violation error messages. Section 6 provides a case study motivating our claims about time savings in deployed ranges, Section 7 reviews related work, and Section 8 concludes this paper.

2. CCER

CCER is an ontology used for specifying operationally relevant and representative cyber range event environment configurations. Ontologies have been used previously to specify cyber event assets, for example by Nodine et al. [10]. The CCER ontology is specified in the Web Ontology Language, OWL [11]. The CCER ontology contains representative and constructive models [1] of configuration of entities in the event environment.

The CCER ontology contains multiple topics. Each topic is a largely self-contained configuration ontology unto itself. The current topics include the following:

- **User:** organizations, teams, and behaviors of users.
- **Application & OS:** applications commonly found on enterprise office computers, such as Microsoft PowerPoint or Outlook, and operating systems. Both the OS and Application topics use Common Platform Enumeration (CPE) strings [12] to describe their contents. CPE strings allow a precise specification of the make, model, and version of the software or platform.
- **Service:** multiple types of services such as DHCP, DNS, file, firewall, or proxy.
- **IP Device:** devices with IP addresses such as computers or devices that support IP-based traffic such as routers.
- **Ethernet Device:** Ethernet devices, and technologies that support network traffic.
- **Physical Location:** the cyber range and the sites where it is located.
- **Visual Location:** the visual location of shape instances in a diagram.
- **Control Plane:** entities such as traffic generators and other assets that control the entities and activities within the Event Plane.
- **Instrument Plane:** data collection services or probes.

In the next section, we provide an overview of our system for creating diagrams, applying rules, and generating errors.

3. DIAGRAMS, RULES, AND ERRORS

Designers describe event environments using a diagram. When developing verification rules, it is important to meaningfully report errors in the diagram to the designers, especially if they are unfamiliar with the rules. Therefore, error or warning statements must be specified within the context of how an environment is represented (in this case, as a diagram). In this section, we provide an overview of our system to design an event environment and generate rule violation error messages.

3.1. Processing an Event Environment Description

Fig. 1 provides an overview of how an event environment description is processed by the CCER tool suite. First, an environment designer creates a diagram to represent an event environment using a visual editor. In CCER, such diagrams are currently created using Microsoft Visio, though other visual editors may be supported in the future. A visual editor presents a user with a palette of shapes, each of which captures the configuration requirements of an entity in the event environment. A *shape* will have a set of fields to which the designer assigns values. The shapes are related via spatial-arrangements described in a visual schema, which we describe in more detail in the next section. When using Microsoft Visio, a CCER Visio Stencil defines the visual schema. A diagram consists of *shape instances* that are instantiations of the shapes with entries in their fields. A shape instance is usually created by dragging a shape into the work area, and then assigning values to its fields.

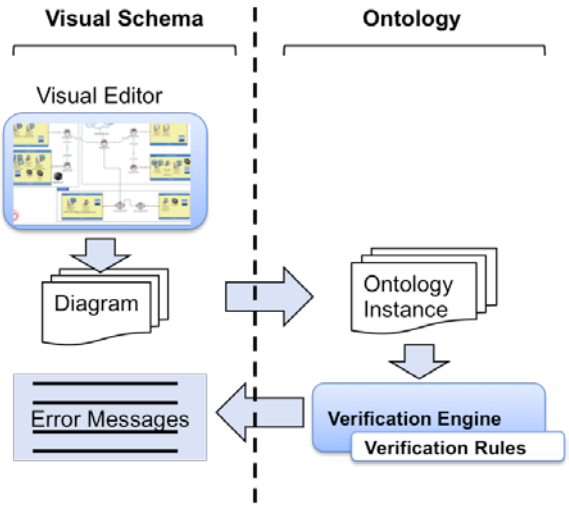

Figure 1: System Overview

An *ontology instance* contains information corresponding to a specific entity in the event environment. For example, in the IP Device topic, the Computer Ontology instance will specify the configuration of a specific Computer. This instance is specified as a *NamedIndividual* [11], `Ent-NS-Client1_CCERID_44_1`, belonging to the Computer OWL Class (Fig. 2). The shape instances in a diagram are translated into CCER ontology instances containing many NamedIndividuals of OWL classes for further processing.

The NamedIndividuals also specify one or more data and object properties. An object property specifies a reference to another NamedIndividual, whereas a data property specifies any value other than such a reference. In RDF, property statements are referred to as triples with the structure <Subject, Property, Object>. In Fig. 2, the NamedIndividual corresponding to the Computer OWL Class is the Subject, and object properties specify the NamedIndividual that is the Object. For example, *hasUser* is an object property with the value `Person_CCERID_1_17` that is a NamedIndividual of a User OWL Class. There are multiple *hasUser* object properties with different such NamedIndividuals as Objects.

Furthermore, the data property *hasInstalledApplication* specifies the application's CPE string. Note that there are multiple such properties with different CPE strings for this Computer instance.

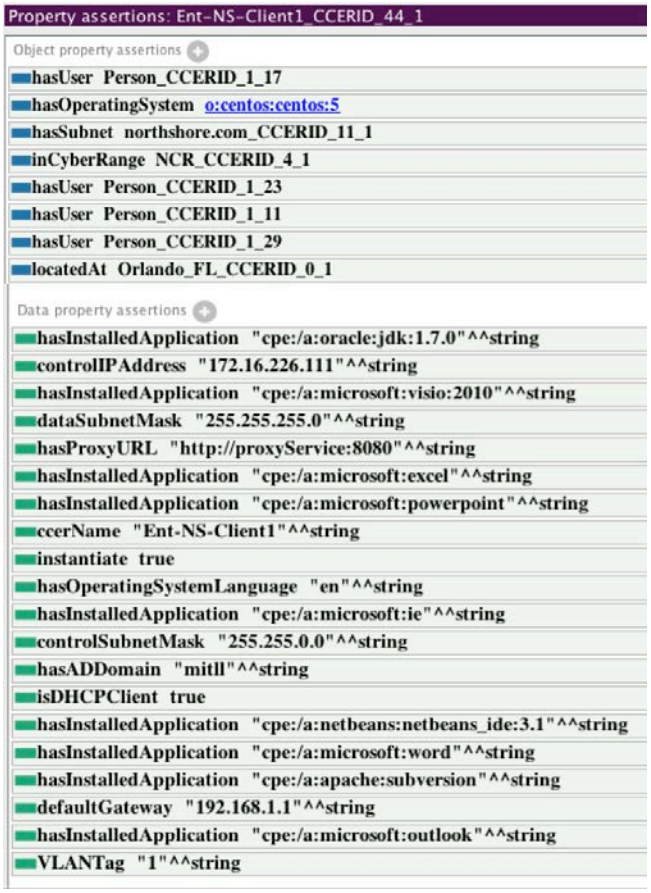

Figure 2: Example Computer Ontology Instance

To summarize, the CCER ontology defines the OWL Classes and any data and object properties, while the ontology instance defines the NamedIndividuals that are members of OWL Classes and the values for the data and object properties of NamedIndividuals. Explaining all of the properties for the Computer instance in Fig. 2 is outside the scope of this paper.

The verification engine processes the ontology instance by applying the verification rules. The verification engine attempts to find all errors, instead of finding the first error and stopping. The rules should be processed over the ontology instances corresponding to both error-free and erroneous diagrams. This requirement imposes much harder requirements in processing diagrams. To prevent verification engine from crashing while processing such diagrams, we do several preprocessing steps. For example, to be able to report Uniqueness Violation error (explained in Section 4), irrespective of whether the values specified in a shape instance are unique or not, unique NamedIndividuals are automatically created for every shape instance in a diagram because RDF ignores duplicate NamedIndividuals [11].

3.2. Visual Schema

Visual schemas describe objects in terms of the physical properties and spatial-arrangements of their components [13]. CCER *visual schema* defines the shapes in diagrams, their fields, the spatial-arrangements and implied semantic relationships among the shapes. We consider a visual schema as *static* if the shapes, fields, default values of the fields, or relationships among the shapes do not change during the creation of a diagram, and as *dynamic* if they do change.

In CCER, the current visual schema is specified using a Visio Stencil [14]. The CCER Visio Stencil is a static visual schema. A Visio Stencil (.vss file) is a collection of shapes associated with a particular Microsoft Office Visio template (.vst file). A CCER Visio Stencil will hold shapes that contain shapes with fields created by the CCER team.

The CCER visual schema specifies the shapes and their fields, and includes the following relationships.

1. **Fields:** The CCER visual schema may specify default values for fields of shapes, identify required and optional fields, or restrict values for a field through a drop down list.
2. **Container:** A Container relationship may be defined between a *Container* shape and a *Containee* shape if and only if some of the values specified in the fields of the shape instance of the Container shape are inherited by the shape instance of the Containee shape. For example, in Fig. 3, the Subnet shape is a Container shape, and its "west.org" instance contains other shape instances such as a Computer, Server, and DNS Service. In this case, the value of a field in west.org may be inherited by these shape instances. Note that a shape can be both a Container and a Containee. For example, the Subnet shape itself may be a Containee shape with respect to another shape, Cyber Range.

Figure 3: Subnet shape is a Container

3. **Connection:** One shape is a Connector to, or is participating in a Connection relationship with, another shape when the *visual connection* of their respective shape instances in a diagram implies that one of the shapes inherits some of the other's fields. Defining and

recognizing when a visual connection is valid is left to the visual editor.

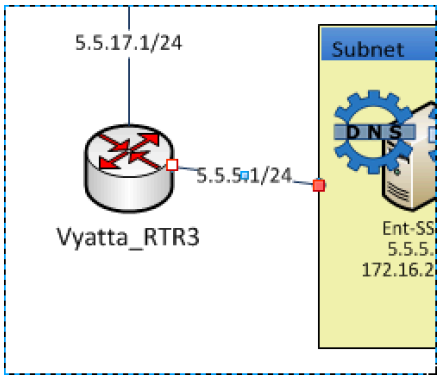

Figure 4: Router Interface shape as Connector

For example, in Fig. 4, the line labeled "5.5.5.1/24" is a Router Interface instance. A Router Interface is a Connector shape. The empty red square on the left end and the solid red square on the right end indicate that the connection is valid. Indeed, the Router Interface is connected to the Router shape instance on the left and the Subnet shape instance on the right. The use of these rectangles to show this connection is specific to Microsoft Visio, and is not part of the visual schema.

4. **Aggregation:** Shape A is an "Aggregator" of shape B if and only if a shape instance of A can be used to specify multiple shape instances of B. In Fig. 5, a ComputerGroup shape instance is shown that specifies 5 Computers in the HostCount field, starting at IPAddress 10.1.2.6, each with the "windows_8" Operating System.

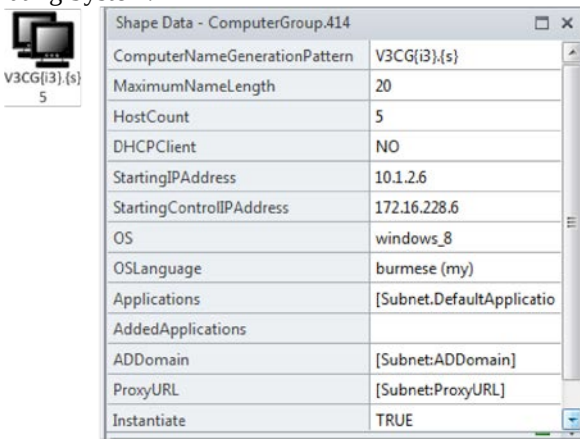

Figure 5: Aggregator Example

5. **Scope:** Shape A is scoped within shape instance B, if the processing of shape instance A's fields is done by analyzing shape instance B, and all other shape instances that are scoped within shape instance B. If a shape instance is not scoped within any other shape instance, then its scope is considered global. The Container relationship mentioned earlier may result in a Scope relationship between the Containee and Container shapes. However, it is not necessary to have a Container relationship for a Scope relationship to exist. For example, a User Type shape instance may specify, in a HostAssignment field, a set of Subnets where users of a specific User Type may need to be created. User Type does not have a Containee relationship to Subnet shape, but the scope of User Type includes the Subnets specified.

A shape may participate in multiple relationships simultaneously, in which case all such relationships are used to calculate the field values of its shape instance. Let us consider the case when a shape participates in Container and Connection relationships simultaneously. For example, the Subnet shape usually participates in both: as a Container, and in a Connection relationship. When a Router Interface connector shape instance has a field with a Default Gateway value, and is connected to a Subnet shape instance, the Subnet shape instance inherits this value, and the Computer shapes within the Subnet shape instance in turn will also inherit this value.

3.3. Diagrams and Error Statements

Error messages, including warning messages, are generated when a verification rule violation is detected. The errors need to be described in terms of the shape instances and fields to make sense to the diagram designer, even though the rule that flags this error is processed over the ontology instance that may have different names and relationships. Errors may be shown visually on a diagram by annotating a shape instance, or through textual error statements that refer to shape instances and the values in their fields. However there are challenges in creating such error messages. We describe some of these challenges below.

Complex Relationships

When the ontology instance and shape instance have a one-to-one correspondence between the shape fields and OWL Class properties, it is easy to specify the error messages in terms of the shape instances, even though the rule is implemented over the ontology instance. However, sometimes fields in a shape instance may not map to a single ontology instance. This is usually the case when some of the relationships described in the previous section are involved. Below are a couple of examples that illustrate this situation.

Container Relationship Example: In this case, errors in an ontology instance may be caused by the absence or presence of values inherited by a shape instance from a Container type shape instance. Consider a Subnet shape that is a Container shape (see Fig. 3). The Host and Server shape instances within a Subnet shape instance inherit the value of the Default Gateway field from the Subnet shape instance. Therefore, an error in the Default Gateway value discovered on a Host shape instance is really caused by the Default Gateway value in the Subnet shape instance, and should be reported as such. In some situations, the shapes that are within a Container shape are allowed to override

the inherited value, and in such situations, the inherited value and the provided value need to be compared prior to generating the error message.

Aggregation Relationship Example: Since an aggregator shape instance (see Fig. 5) describes multiple ontology instance elements and properties using a single shape instance, when an error is detected in one generated ontology instance, then all such generated instances will have the same error. To prevent a deluge of the same error for all such instances, the error messages should be limited to just one.

4. TYPES OF VERIFICATION ERRORS

CCER's rules were developed organically over two years, motivated by the needs of actual cyber range events. We have analyzed the verification rules in terms of the types of errors they describe. The following may serve as a checklist for deriving rules.

1. **Invalid Format:** The value entered in a visual field does not match the value's required format.
2. **Unspecified Value:** A required value is not specified in a field of the shape.
3. **Inconsistent Value:** A specified value is not consistent with another value elsewhere in the diagram.
4. **Out-of-range Value:** The specified value is out of the contiguous range or the set of allowed values for the value. The range can be directly or indirectly specified. An indirect specification occurs when the range for a field value is implied due to the value of a field attribute elsewhere in the model.
5. **Uniqueness Violation:** When the same shape instances or corresponding ontology elements require uniqueness in one or more field values within a specified scope, and this uniqueness is violated, this error occurs. In our experience, the most common cause of this error is "copy-and-pasting" shape instances in a diagram.
6. **Non-existent Reference (Direct):** A reference specified in a field does not exist in the diagram.
7. **Non-existent Reference (Indirect):** A value specified for a field does not match a related field in any other shape instance.
8. **Unspecified Reference:** A required reference is not specified in a field of a shape.
9. **Inconsistent Reference:** A specified reference is not consistent with another specified reference elsewhere in the model. This type of error indicates where the visual schema may be made better by automatic inference.
10. **Singleton Violation:** When only one shape instance is permitted within a given scope, and this rule is violated, this error occurs.
11. **Generational Insufficiency:** In CCER, values for some OWL ontology instance properties are calculated from the specified field values or references in shapes. When these values are either out of range or are not generated, then this error is flagged.
12. **Abnormal Specification:** Sometimes, the specified diagram has no model-violating errors. However, it is not good practice, or is abnormal, to make such specifications. We, therefore, classify these errors as "abnormal."

In the above classification of errors, we avoid "Incorrect Value" as a type of error. Instead, we have further refined this type of error into other types: Unspecified Value, Out-of-range Value, Inconsistent Value, Uniqueness Violation, and Generational Insufficiency. Similarly, instead of "Incorrect Reference," as a type of error, we further refined that into other types: Non-Existent Reference (Direct/Indirect), Inconsistent Reference, and Singleton Violation.

In the next section, we discuss both the methodology of verification rule development and corresponding error statement specification using CCER with examples.

5. METHODOLOGY FOR RULE DEVELOPMENT

While developing verification rules for elements within the various ontology topics, over time, we observed that we were repeatedly following the same steps. Developing the verification rules requires a precise model of the configuration of the components or concepts, a specification of the OWL Classes, and relevant object and data properties. We describe these steps below, and also illustrate with an example based on a DHCP service.

Step 1: Develop a written description of the configuration of the entities in sufficient detail. Note that there may be many entities that interact in a model description, and some of the existing entities may need changes due to interactions with the new configurations.

Example: The DHCP Service will need an IPAddress Range specification that will be allocated to Hosts, and IPAddress Exceptions in the IPAddress Range. The DHCP Service will need to specify the server on which it will run (DHCPServiceName). The IPAddress Range should be identified as public or private. Each entity that can have an IPAddress in the Event Plane must specify whether it is a DHCP Client. Currently, only dynamic DHCP address allocation is permitted. Also, DHCP Service is currently not permitted on a Router.

Step 2: Describe the shapes, their attributes, and constraints such as available choices that are required to generate the corresponding ontology instance.

Example: See Fig. 6. The ServerName specifies the server on which this DHCP Service will run. The StartingDataIPAddress and the EndingDataIPAddress specify the IPAddress Range for allocation. The IPAddress Exceptions specify the IPAddress exceptions separated by

a semicolon. The StartingControlIPAddress and the EndingControlIPAddress specify the range of ControlIPAddresses. The ControlIPAddressExceptions specify the exceptions separated by a semicolon. Finally, the IPAddressSpace allows a drop down list of Private or Public (with Private being the default).

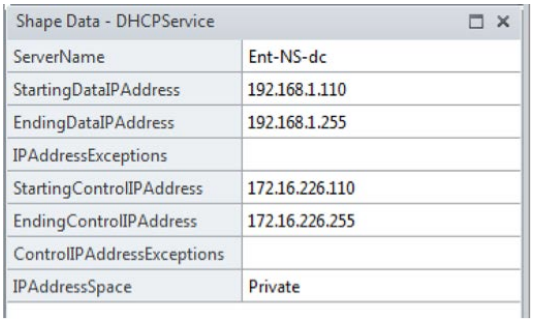

Figure 6: Fields

Step 3: Describe in detail the OWL Classes, data properties, and object properties of the ontology corresponding to the configuration. Also, create a sample ontology instance for the OWL Classes with sample object and data properties.

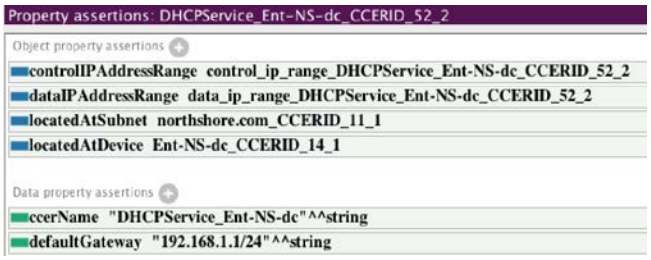

Figure 7: An Ontology Instance for DHCP Service

Example: Fig. 7 describes an ontology instance of an OWL Class for the DHCP Service with the associated object and data properties. Note that the OWL Classes corresponding to the "object" of object properties also need to be created if they do not already exist.

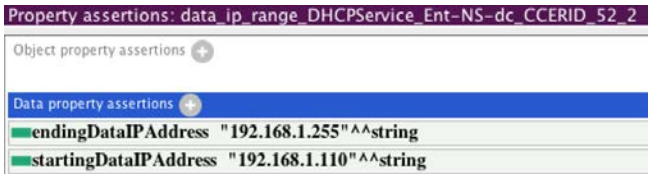

Figure 8: IPAddressRange Instance

The instances (NamedIndividuals) of the IPAddressRange OWL Class are used in the *dataIPAddressRange* and *controlIPAddressRange* object properties (Fig. 8).

Step 4: Define the verification rules and error statements. In this step, the shapes in the fields are evaluated for all of the error categories discussed in Section 4.

Example: The example shown in Fig. 9 shows one of the DHCP Rules and the corresponding error message. In the error message, the phrase "DHCP Service:" refers to the shape concerned, and the phrase starting with "$DHCPService" refers to the corresponding ontology instance. If the ExceptionIPAddresses specified by the users needed to be shown in the error message, those would need to be specified with both the ":" and "$" qualifiers.

Rule ID	DHCP3
Error Category	Invalid Format
Shape	DHCP Service
Rule	DHCP Service Exception IP Addresses are specified in correct format
Error Message	DHCP Service: $DHCPService Exception IP Addresses have incorrect format

Figure 9: Rule and Error Message Example

6. CASE STUDY

A case study on the use of the CCER tool suite will be helpful in understanding where time is spent when designing a new environment. Fig. 10 summarizes the time spent in designing a moderately sized event environment with 5 subnets, a simulated Internet, 3 routers, and 5 switches, with approximately 100 computers and users and 25 services. The x-axis is the actual time spent in hours on the activities shown on the y-axis. Training included both in-person training and answering questions via email and phone.

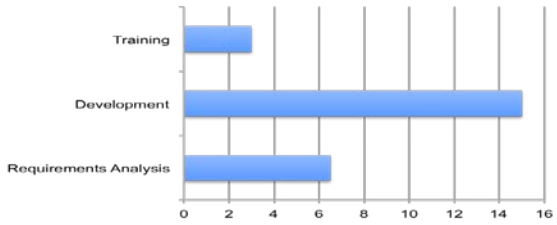

Figure 10: Time Statistics

Fig. 10 shows that the development activity took most of the time, and the requirements analysis required less than half as much time. The requirement analysis activity included discussions on specific missing details, including those that caused verification errors. In this particular case's requirements analysis, an initial sketch of a diagram was drawn prior to working with the CCER tool suite, and a record of the time spent drawing that diagram was not available for inclusion in our statistics. Over 300 errors were reported and corrected in the development phase, but the deployment phase did not get delayed due to misconfiguration of the entities in the diagram. These proportions are consistent with other cases we have worked on.

Our system currently has over 125 verification rules. In many cases, we are able to create error messages that directly refer to the erroneous shape instances in the diagram. Frequent users of the system are able to make sense of even cryptic error messages. We find that both new

users and frequent users are occasionally surprised by the errors shown, though for different reasons. New users are surprised because of their unfamiliarity with the errors, and frequent users are surprised because they did not expect to make such simple errors. Most users took less than a day to become comfortable developing new diagrams, understanding the error messages, and correcting errors for commonly used shapes.

7. RELATED WORK

In previous work applying model based verification and validation for software configuration Cordero et al. [15] built a software development environment that uses to support the development and verification of on-board software for spacecraft. Tanizaki et al. [16] use a SAT solver approach to model and check the consistency of configuration files and discover syntactic and semantic errors. Quinton et al. [17] have established an architecture based on feature models and ontologies to describe and model cloud computing systems, reducing variability when working with multi-cloud configurations.

In the field of firewall configuration and policies, current approaches to model based verification and validation include work by Brucker et al. [18], who provide a formal model of stateful and stateless firewalls, and a framework that tests actual firewalls using that model. Moussa et al. formalize the process of verifying consistency among a set of firewalls with a global security policy [9]. Adão et al. [19] present a tool for converting abstract firewall models into concrete configurations for the Netfilter networking framework in Linux. Windmüller [7] uses a configuration that combines a model checker, a graphical modeling environment, and external tools to simplify the process of testing and validation of firewall setups for end users. Industry efforts such as Cisco's CLI [20] have focused on specific entities such as Routers. Finally, Anderson et al. [6] report on an online network validation tool for Emulab.

However, the approaches mentioned above do not fully address the needs for configuring a rich set of entities within an event environment. These approaches also are not designed to be sufficiently extensible and flexible to support the addition of new domains, nor do they address the problem of explaining errors in a way that a novice designer can understand.

We hope to evaluate the models developed in such works for future cyber range event environments. For example, cyber ranges often attempt to replicate critical infrastructure networks—such as those belonging to a transportation system, a hospital, or an industrial plant—that have both cyber and physical components. This is in the tradition of Tang et al. [21], who extended the use of Modelica, a modeling language to describe complex physical systems, into the realm of cyber-physical systems modeling.

8. CONCLUSIONS

In this paper, we have presented a model based verification approach employed in the CCER tool suite for use by cyber range event environment designers. We identified the key role a visual schema plays in generating appropriate error messages when verification rules are violated. We have presented a categorization of configuration errors, and described a methodology for creation of rules and error messages. This methodology is being used to create additional verification rules for new domains. We have successfully used the CCER tool suite to create cyber range environments of varying complexity and scale for several events.

We still have much work to do to improve our systems. One area for improvement is reducing the number of error messages corresponding to the same user mistake. Another area is the evaluation of the impact of a dynamic visual schema on reducing the time involved in detection and fixing errors. Making the error messages more intuitive for designers is another area of future improvement.

We continue to expand our configuration ontologies, write new rules and error messages using the methodology described in this paper. We also plan a performance analysis of the rules verification engine to study how performance is affected by the size of the diagrams.

9. ACKNOWLEDGEMENTS

Ritesh Patel and Serge Vilvovsky made substantial contributions to the implementation of the CCER tool suite, including the rule verification system. We also thank Antonio Roque and Christine Cunningham for their comments.

This work is sponsored by the Test Resource Management Center under Air Force Contract FA8721-05-C-0002. Opinions, interpretations, conclusions and recommendations are those of the author and are not necessarily endorsed by the United States Government.

REFERENCES

[1] Suresh K. Damodaran and Jerry M. Couretas, "Cyber Modeling & Simulation for Cyber-Range Events," in *Summer Simulation Multi-Conference*, Chicago, IL, USA, 2015.

[2] Z. Yin, X.,Zheng, J., Zhou, Y. Ma, and L.N., Pasupathy, S. Bairavasundaram, "An Empirical Study on Configuration Errors in Commercial and Open Source Systems," in *SOSP*, Cascais, Portugal, 2011.

[3] Sanjai Narain, Sharad Malik, and Ehab Al-Shaer, "Towards Eliminating Configuration Errors in Cyber Infrastructure," in *4th IEEE Symposium on Configuration Analytics and Automation*, Arlington, VA, 2011.

[4] Sanjai Narain, Rajesh Talpade, and Gary Levin, "Network Configuration Validation," in *Guide to*

Reliable Internet Services and Applications. London: Springer-Verlag, 2010.

[5] Sanjai Narain, "Network Configuration Management via Model Finding," in *19th Large Installation System Administration Conference*, San Diego, 2005, pp. 155-168.

[6] David S. Anderson, Mike Hibler, Leigh Stoller, Tim Stack, and Jay Lepreau, "Automatic Online Validation of Network Configuration in the Emulab Network Testbed," *3rd IEEE International Conference on Autonomic Computing*, 2006.

[7] Stephan Windmüller, "Simplifying Firewall Setups by Using Offline Validation," *Journal of Integrated Design and Process Science*, vol. 17, no. 3, pp. 59-69, 2013.

[8] Ehab Al-Shaer and Saeed Al-Haj, "FlowChecker: Configuration Analysis and Verification of Federated OpenFlow Infrastructures," in *SafeConfig '10*, Chicago, 2010, pp. 37-44.

[9] Majda Moussa, Hakima Ould-Slimane, Hanifa Boucheneb, and Steven Chamberland, "A Formal Framework for Verifying Inter-Firewalls Consistency," in *19th IEEE Symposium on Computers and Communication*, Piscataway, 2014.

[10] M Nodine, R. Grimshaw, P. Haglich, S. Wilder, and B Lyles, "Computational Asset De-scription for Cyber Experiment Support using OWL ," in *Proc. of Fifth IEEE International Conference on Semantic Computing (ICSC)*, 2011, pp. 110-117.

[11] W3C. (2012, December) World Wide Web Consortium (W3C). [Online]. http://www.w3.org/TR/owl-overview/

[12] National Institute of Standards and Technology. (2014, January) The Security Content Automation Protocol (SCAP) - NIST. [Online]. http://scap.nist.gov/specifications/cpe/

[13] Risto Miikkulainen and Wee Kheng Leow, "Visual Schemas in Object Recognition and Scene Analysis," in *The Handbook of Brain Theory and Neural Networks*. Cambridge, MA: MIT Press, 1995, pp. 1029-1031.

[14] Microsoft. (2015) Support - support.office.com. [Online]. https://support.office.com/en-au/article/A-beginner-s-guide-to-Visio-2010-88d2a308-7283-4981-839d-86e2aca8c456

[15] Federico Cordero et al., "A Cost-Effective Software Development and Validation Environment and Approach for LEON Based Satellite & Payload Subsystems," in *5th International Conference on Recent Advances in Space Technologies*, Piscataway, 2011.

[16] Hiroaki Tanizaki, Toshiaki Aoki, and Takuya Katayama, "A Variability Management Method for Software Configuration Files," in *The 24th International Conference on Software Engineering and Knowledge Engineering*, Redwood City, 2012.

[17] Clément Quinton, Nicolas Haderer, Romain Rouvoy, and Laurence Duchien, "Towards Multi-Cloud Configurations Using Feature Models and Ontologies," in *International Workshop on Multi-Cloud Applications and Federated Clouds*, Prague, 2013.

[18] Achim D. Brucker, Lukas Brügger, and Burkhart Wolff, "Formal Firewall Conformance Testing: An Application of Test and Proof Techniques," in *Software Testing, Verification and Reliability*, 2015, pp. 34-71.

[19] P. Adão, C. Bozzato, G. Dei Rossi, R. Focardi, and F. L. Luccio, "Mignis: A Semantic Based Tool for Firewall Configuration," in *27th Computer Security Foundations Symposium*, Los Alamitos, CA, 2014.

[20] Cisco Systems, Inc. Cisco IOS Configuration Fundamentals Configuration Guide Release 12.2. [Online]. http://www.cisco.com/c/en/us/td/docs/ios/12_2/configfun/configuration/guide/ffun_c.pdf

[21] Junjie Tang, Jianjun Zhao, Jianwan Ding, Liping Chen, and Gang Xie, "Cyber-Physical Systems Modeling Method Based on Modelica," in *International Conference on Software Security and Reliability Companion*, Los Alamitos, 2012.